MIT/LCS/TR-413

DIVERSITY-BASED REDUCTION OF POWER REQUIREMENTS

Robert Elias Schapire

May 1988

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TR-413

DIVERSITY-BASED INFERENCE OF FINITE AUTOMATA

Robert Elias Schapire

May 1988

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Diversity-Based Inference of Finite Automata

by

Robert Elias Schapire

Sc.B., Mathematics and Computer Science
Brown University
(1986)

submitted to the
Department of Electrical Engineering and Computer Science
in partial fulfillment
of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology
May 1988

Signature of Author _____
Department of Electrical Engineering and Computer Science
April 21, 1988

Certified by _____
Ronald L. Rivest
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Diversity-Based Inference of Finite Automata

by

Robert Elias Schapire

Submitted to the Department of Electrical Engineering and Computer Science
on April 21, 1988 in partial fulfillment of the requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

We present a new procedure for inferring the structure of a finite-state automaton (FSA) from its input/output behavior, using access to the automaton to perform experiments.

Our procedure uses a new representation for FSA's, based on the notion of equivalence between *tests*. We call the number of such equivalence classes the *diversity* of the automaton; the diversity may be as small as the logarithm of the number of states of the automaton. For the special class of *permutation automata*, we show that our inference procedure runs in time polynomial in the diversity and $\log(\frac{1}{\epsilon})$, where ϵ is a given upper bound on the probability that our procedure returns an incorrect result. (Since our procedure uses randomization to perform experiments, there is a certain controllable chance that it will return an erroneous result.) We also discuss techniques for handling more general automata.

We present evidence for the practical efficiency of our approach. For example, our procedure is able to infer the structure of an automaton based on Rubik's Cube (which has approximately 10^{19} states) in about 2 minutes on a DEC MicroVax. This automaton is many orders of magnitude larger than possible with previous techniques, which would require time proportional at least to the number of global states. (Note that in this example, only a small fraction (10^{-14}) of the global states were even visited.)

Finally, we present a new procedure for inferring automata of a special type in which the global state is composed of a vector of binary local state variables, all of which are observable (or *visible*) to the experimenter. Our inference procedure runs provably in time polynomial in the size of this vector (which happens to be the diversity of the automaton), even though the global state space may be exponentially larger. The procedure plans and executes experiments on the unknown automaton; we show that the number of input symbols given to the automaton during this process is (to within a constant factor) the best possible.

Portions of this thesis are joint work with Ronald Rivest.

Thesis Supervisor: Ronald L. Rivest

Title: Professor of Computer Science and Engineering

Keywords: Learning, Automata, Induction, Inference of Finite Automata

This paper prepared with support from NSF grant DCR-8607494, ARO Grant DAAL03-86-K-0171, and a grant from the Siemens Corporation.

Acknowledgements

First and foremost, I wish to thank my advisor Ron Rivest for his help, patience and encouragement, and for the many things I have learned working with him.

For their contributions to particular results included in this thesis, I thank Dana Angluin, Satish Rao and Neal Young. I also thank Bob Sloan for his comments on an earlier draft, as well as Be Hubbard, Jon Riecke, my officemates, my parents, and especially Roberta Sloan for their personal support.

Finally, I gratefully acknowledge the financial support I received from MIT, Siemens Corporation, and the Center for Intelligent Control Systems.

Contents

Chapter 1

Introduction

We address the problem of inferring a description of a deterministic finite-state automaton from its input/output behavior.

Our motivation is the "artificial intelligence" problem of identifying an environment by experimentation. We imagine a robot wandering around in an unknown environment, whose characteristics must be discovered. Such an environment need not be deterministic, or even finite-state, so the approach suggested here is only a beginning on the more general problem.

In line with our motivation, our inference procedures experiment with the automaton to gather information.

A unique and valuable feature of our procedures is that they do *not* need to have the automaton "reset" to some start state or "backed-up" to a previous state; the data-gathering is one continuous experiment (as in real life).

Our procedures are practical; their time and memory requirements are quite reasonable. For example, our procedures do not need to store the entire observed input/output history.

In Chapters 2 and 3, we present a new representation of finite automata based on the notion of test equivalence. We present and prove the effectiveness of a probabilistic algorithm for inferring permutation automata. We also discuss possible techniques for handling more general automata, and give some preliminary experimental results.

In Chapter 4, we extend the work of the preceding chapters focusing on one aspect of the inference problem, namely, that of planning experiments for gathering information.

Portions of this thesis were previously described in [14,15].

1.1 Previous Work

For a fascinating discussion of the problem of inferring an environment from experience, the reader is encouraged to read Drescher [5], whose approach is based on the principles of Piaget.

Kohavi [12] gives a fine introduction to the theory of finite-state automata, as do Hartmanis and Stearns [11]. An excellent overview of the entire field of inductive inference is

given by Angluin and Smith [4].

The problem of inferring a finite-state automaton from its input/output behavior has a long history. In [9], Gold presents a number of recursion theoretic results concerning several language classes, including the regular languages. Gold looks first at the problem of identifying a language when given a particular presentation of the language. In one case, the learner is provided with an infinite stream in arbitrary order of all strings in the language. In the second case, the learner is given an infinite stream of all finite strings generated by the alphabet, each string labeled as to whether it is or is not an element of the language. In his model, the learner guesses the identity of the language after each example, and is said to learn the language "in the limit" if, after a finite number of examples, the learner's guesses converge to the right answer and the learner never again changes its guess. Gold shows regular languages can be learned in the limit in the second case described above, but not in the first.

In the same paper, Gold describes the problem of "black box" identification, closely related to the particular problem that we are here addressing. In this situation, the learner is able to experiment with an unknown black box. At each time step, the learner supplies the black box with one of a finite number of input symbols and the black box in turn outputs an output symbol calculated as a function of the input symbols provided to it up to that point in time. Gold shows that if the black box is a finite automaton, then it can be identified in the limit. Note that Gold's results were recursion theoretic and did not address the time complexity of any of these problems.

In [10], Gold examines more closely the problem of inferring a black box finite automaton. In this paper, Gold assumes that the experimenter has available to it a means of resetting the automaton to some initial state. He describes how the automaton can be identified in the limit, how experiments can be efficiently planned, and how the automaton can be identified in a finite amount of time if the learner is given beforehand the number of states of the automaton.

Angluin [2] elaborates this algorithm to show how to infer an automaton with active experimentation. In her model, the learner has a "minimally adequate teacher" who can answer two kinds of queries: First, the teacher will tell the learner whether any particular string is a member of the unknown language. Second, the teacher is able to supply the learner with a counterexample to an incorrect conjecture of the automaton's identity. Angluin shows that the number of queries required by her algorithm to correctly identify the unknown automaton is only polynomial in the number of states of the automaton.

Angluin [3] and Gold [8] prove that finding an automaton of n states or less agreeing with a given sample of input/output pairs is NP-complete. Note that in this situation the

inference algorithm does not have access to the automaton—the input/output pairs are given and the learner is not able to experiment with the automaton it is trying to identify.

Finally, Angluin [1] shows how to infer in polynomial time a special-class of finite-state automata, called "k-reversible" automata, from a sample of input/output behavior. Later, we will give special consideration to the class of permutation automata of which the zero-reversible automata are a subclass.

Chapter 2

A New Representation of Finite Automata

2.1 Automata and Environments

Our definition of a finite-state automaton is a generalization of the usual Moore automaton. [12]. (Our approach generalizes to handle Mealy automata; however, we find Moore automata more natural.)

Definition 1 *A* finite-state automaton $\mathcal{E}$ *is a* 6*-tuple* $(Q, B, P, q_0, \delta, \gamma)$ *where*

- Q *is a finite nonempty set of* states.
- B *is a finite nonempty set of* input symbols, *also called* basic actions.
- P *is a finite nonempty set of* predicate symbols, *also called* sensations.
- q_0, *a member of* Q, *is the* initial state.
- δ *is a function from* $Q \times B$ *into* Q; δ *is called the* next-state function.
- γ *is a function from* $Q \times P$ *into* $\{\mathbf{true}, \mathbf{false}\}$.

When P only contains a single predicate (e.g. **accept**), we have the standard definition of a Moore automaton. We allow multiple predicates to correspond to the notion of a robot having multiple sensations in a given state of the environment.

We assume henceforth that we are dealing with a particular finite-state automaton $\mathcal{E} = (Q, B, P, q_0, \delta, \gamma)$, which we call the *environment* of the learning procedure.

We say that $\mathcal{E}$ is a *permutation environment* if for each $b \in B$, the function $\delta(\cdot, b)$ is a permutation of Q.

We let $A = B^*$ denote the set of all sequences of zero or more basic actions in the environment $\mathcal{E}$; A is the set of *actions* possible in the environment $\mathcal{E}$, including the *null action* λ.

If q is a state in Q, and $a = b_1 b_2 \ldots b_n$ is an action in A, we let $qa = qb_1 b_2 \ldots b_n$ denote the state resulting from applying action a to state q:

$$qa = \delta(\ldots \delta(\delta(q, b_1), b_2) \ldots, b_n). \tag{2.1}$$

(The basic actions are performed in the order $b_1, b_2, \ldots, b_n$.) Similarly, if q is a state and p is a predicate, we let $qp = \gamma(q, p)$ denote the result of applying predicate p to state q.

We say that $\mathcal{E}$ is *strongly connected* if

$$(\forall q \in Q)(\forall r \in Q)(\exists a \in A)qa = r. \tag{2.2}$$

We do not assume that $\mathcal{E}$ is strongly connected in our general discussion of automata and diversity.

However, when we describe our inference procedure, we will make this assumption with little loss of generality: If $\mathcal{E}$ is not strongly connected, then an experimenting inference procedure, having no "reset" operation, will sooner or later fall into a strongly connected component of the state space from which it cannot escape, and so will have to be content thereafter learning only about that component.

2.2 Tests

A *test* is an element of AP, that is, an action followed by a predicate. We let T denote the set of tests AP. We say that a test $t = ap$ *succeeds at state* q if $qt = q(ap) = qap = (qa)p$ is **true**. Otherwise we say that t *fails at* q. The *length* $|t|$ of a test t is the number of basic actions and predicates it contains.

We say that $\mathcal{E}$ is *reduced* if every pair of states can be distinguished by executing some test:

$$(\forall q \in Q)(\forall r \in Q)(q \neq r \Rightarrow (\exists t \in T)qt \neq rt) \tag{2.3}$$

We assume henceforth that $\mathcal{E}$ is reduced.

We say that a robot has a *perfect model* of its environment if it can predict perfectly what sensations would result from any desired sequence of basic actions, that is, if it knows the value of every test in the current state. The goal of our inference procedures is to build a perfect model of the given environment.

2.3 Equivalence of Tests and Diversity

A central notion in our development is that of *test equivalence.*

We say that tests t_1 and t_2 are *equivalent*, written $t_1 \equiv t_2$, if

$$(\forall q \in Q)(qt_1 = qt_2); \tag{2.4}$$

that is, from any state the two tests yield the same result.

The equivalence relation on tests partitions the set T of tests into equivalence classes. The equivalence class containing a test t will be denoted $[t]$.

The *diversity* of the environment $\mathcal{E}$, denoted $D(\mathcal{E})$, is the number of equivalence classes of tests of $\mathcal{E}$:

$$D(\mathcal{E}) = |\{[t] \mid t \in T\}|. \tag{2.5}$$

The following theorem demonstrates that the diversity of a finite-state automaton is always finite, but is only loosely related to the *size* (i.e. number of states) of the automaton.

Theorem 1 *For any reduced finite-state automaton* $\mathcal{E} = (Q, B, P, q_0, \delta, \gamma)$,

$$\lg(|Q|) \leq D(\mathcal{E}) \leq 2^{|Q|}.$$

Proof: The first inequality $\lg(|Q|) \leq D(\mathcal{E})$, or equivalently $|Q| \leq 2^{D(\mathcal{E})}$, holds because a state is uniquely identified by the set of (equivalence classes of) tests which are true at that state, since $\mathcal{E}$ is reduced. The second inequality holds because the equivalence class that a test belongs to is uniquely defined by the set of states at which that test succeeds. ■

Theorem 2 *The lower and upper bounds on* $D(\mathcal{E})$ *given in Theorem 1 are best possible.*

Proof: For the lower bound, consider an environment where the states are n-bit words, and, for $1 \leq i \leq n$, there is a predicate p_i which tests whether the i-th bit is one. The set B consists of a single action, which is the identity operation (no state change). Then $D(\mathcal{E}) = n$ but $|Q| = 2^n$. Note that the state space in this example is unconnected.

For the upper bound, consider an automaton whose states are represented by an element $\mathbf{x}$ which is either an n-bit vector $(x_1, \ldots, x_n)$ or the special value **hit**; there are $1+2^n$ states. The only predicate tests whether $\mathbf{x} =$ **hit**. The following actions are available:

- For each $i \in \{1, \ldots, n\}$, an action which flips x_i if $\mathbf{x} \neq$ **hit**, and leaves $\mathbf{x}$ alone otherwise.
- An action which sets $\mathbf{x}$ to **hit** if $\mathbf{x}$ is the all-zero vector 0^n, and leaves $\mathbf{x}$ alone otherwise.

Using these actions, for any subset X of the n-bit vectors, it is possible to construct a test which is **true** if and only if the initial state begins with $\mathbf{x} \in X$ or $\mathbf{x} =$ **hit** initially. (Selective complementation can bring $\mathbf{x}$ into the all-zero state iff it was originally in some particular n-bit state $\mathbf{y}$; this state can then be transformed to **hit**, otherwise the original state can be restored by undoing the selective complementation. This can be repeated for each $\mathbf{y} \in X$.) Actually, this environment only comes within a factor of two of the upper bound; its diversity is $2^{|Q|-1}$.

However, the following alternative environment does achieve the upper bound, although its set of basic actions is enormous. The environment consists of n states numbered 0

through $n-1$, and has a single predicate p which succeeds only at state 0. For each subset X of the states, there is an action b_X which moves state x to state 0 if $x \in X$, or to state 1 otherwise. Thus, the test $b_X p$ is **true** iff we are in one of the states in X. Hence, $D(\mathcal{E}) = 2^{|Q|}$. ■

We propose that the notion of diversity is more suitable than that of size for many natural applications. To support this viewpoint, we will demonstrate that *there exists a natural encoding of a finite-state automaton, whose size is polynomial in the diversity of the automaton.* Furthermore, it is straightforward to use this representation to simulate the behavior of the automaton.

2.4 The Update Graph

As a convenient means of representing the test classes, we may build a directed graph in which each vertex is an equivalence class, and an edge labeled $b \in B$ is directed from test class $[t]$ to $[t']$ iff $t \equiv bt'$. We call this the *update graph* of the environment.

Since there is one vertex for each equivalence class, the size of the update graph is precisely the diversity of $\mathcal{E}$. Note that, for $b \in B$, every vertex has exactly one b-edge directed into it, since if $t \equiv t'$ then $bt \equiv bt'$.

Also, for any test $t = ap$ where p is a predicate and $a = b_1 b_2 \ldots b_n$ is a sequence of basic actions, there is a path in the update graph along which vertex $[p]$ can be reached from $[t]$ by following the edges labeled $b_1, b_2, \ldots b_n$. Put another way, we can find t's equivalence class in the update graph by tracing backwards from $[p]$ along the unique path $b_n, \ldots, b_1$.

We associate with each vertex $[t]$ the value of t at the current state q. (This value is well defined since if $t \equiv t'$ then by definition of equivalence, $qt = qt'$.) When action b is executed, the test $[t']$ gets its value from $[t]$, where $t \equiv bt'$, yielding the new value of each test in state qb. Thus, the update graph may be used to simulate the automaton, as we prove in the following theorem.

2.5 The Simulation Theorem

Theorem 3 *To simulate $\mathcal{E}$ it suffices to know:*

1. *The update graph.*
2. *For each equivalence class $[t]$, the value qt at the current state q.*

Proof: Suppose the automaton moves from state q to state qb, for some $b \in B$. We need to compute $(qb)t = q(bt)$ for each equivalence class $[t]$. However, the test bt belongs to that

(unique) equivalence class $[s]$ for which an edge labeled b is directed from $[s]$ to $[t]$ in the update graph. By assumption, we know qs; this is the desired value of $(qb)t$. ■

2.6 Simple Assignment Automata

We may regard the test equivalence classes as (local) *state variables* each of which is updated under the execution of some basic action with the value of one other (or the same) variable. We call such a structure a *simple assignment automaton* (SAA). The output of an SAA consists of the current values of one or more its variables—in this case the equivalence classes of the predicates.

If we regard the current state of an SAA as the assignment of values to all the variables, then it is clear that every SAA is deterministic and finite state, and so can be simulated by some FSA. Conversely, our construction and the simulation theorem show that every FSA can be simulated by some SAA (the one we have constructed is the smallest such SAA). Thus, we have proved:

Theorem 4 *Every SAA can be simulated by an FSA, and every FSA can be simulated by an SAA.*

2.7 Characterizing Diversity and the Update Graph

Neal Young and Dana Angluin have pointed out the following relationship between the update graph of an environment with a single predicate, and the original automaton:

Let $\mathcal{E}$ be an environment with a single predicate, $(Q, B, \{p\}, q_0, \delta, \gamma)$, and let $\mathcal{E}' = (Q', B, \{p'\}, q_0', \delta', \gamma')$ be defined as follows:

- $Q' = \{[t] \mid t \in T\}$
- $q_0' = [p]$
- $\delta'([t], b) = [bt]$, for $[t] \in Q', b \in B$
- $\gamma'([t], p') = q_0 t$, for $[t] \in Q'$.

In this construction, Q' is just the vertex set of $\mathcal{E}$'s update graph so that $|Q'| = D(\mathcal{E})$. Furthermore, by the definition of δ', we see that the transition graph of $\mathcal{E}'$ is exactly this update graph with all of the edges reversed in direction.

Theorem 5 *Let $\mathcal{E}$ and $\mathcal{E}'$ be as described above. Then for any action $a \in A$, $q_0 ap = q_0' a^R p'$ where a^R is the reverse of a.*

Proof: Let $a = b_1 \ldots b_n$, where each $b_i \in B$. Then by the definition of δ', we have:

$$\begin{aligned} q'_0 a^R &= [p] b_n b_{n-1} b_{n-2} \ldots b_1 \\ &= [b_n p] b_{n-1} b_{n-2} \ldots b_1 \\ &= [b_{n-1} b_n p] b_{n-2} \ldots b_1 \\ &\vdots \\ &= [b_1 \ldots b_n p] \\ &= [ap]. \end{aligned}$$

Thus, $q'_0 a^R p' = \gamma'(q'_0 a^R, p') = \gamma'([ap], p') = q_0 ap$. ■

The *language* $L(\mathcal{E})$ accepted by automaton $\mathcal{E}$ is the set of actions $a \in A$ which move $\mathcal{E}$ from its starting state to an "accepting" state in which the the environment's only predicate is **true**. That is, $L(\mathcal{E}) = \{a \in A \mid q_0 ap = \textbf{true}\}$. Theorem 5 shows that the diversity of $\mathcal{E}$ is exactly the state size of the minimum FSA which accepts the reverse of $L(\mathcal{E})$.

When $\mathcal{E} = (Q, B, \{p\}, q_0, \delta, \gamma)$ is a permutation environment with a single predicate, the diversity and update graph can be characterized in a different manner. In this case, the set of basic actions generates a permutation group G on the states of $\mathcal{E}$. Let H be the subgroup of G which stabilizes the accepting states of $\mathcal{E}$. That is, H consists of those group elements a of G such that $qp = qap$ for all $q \in Q$. (Equivalently, G is the permutation group on the test equivalence classes of $\mathcal{E}$, and H is the subgroup of G which stabilizes $[p]$.)

We define the *left coset graph of* H as follows: The vertices of the graph are the left cosets of H, and an edge labeled b is directed from aH to $a'H$ iff $aH = ba'H$.

Then the following theorem shows that the diversity of $\mathcal{E}$ is exactly the index of H in G:

Theorem 6 *The update graph of $\mathcal{E}$ is isomorphic to the left coset graph of H.*

Proof: For any two tests xp and yp, we have:

$$\begin{aligned} xp \equiv yp &\Leftrightarrow y^{-1} xp \equiv p \\ &\Leftrightarrow (\forall q \in Q) q y^{-1} xp = qp \\ &\Leftrightarrow y^{-1} x \in H \\ &\Leftrightarrow x \in yH \\ &\Leftrightarrow xH = yH. \end{aligned}$$

■

The generalization of both these characterizations to environments with multiple predicates is straightforward.

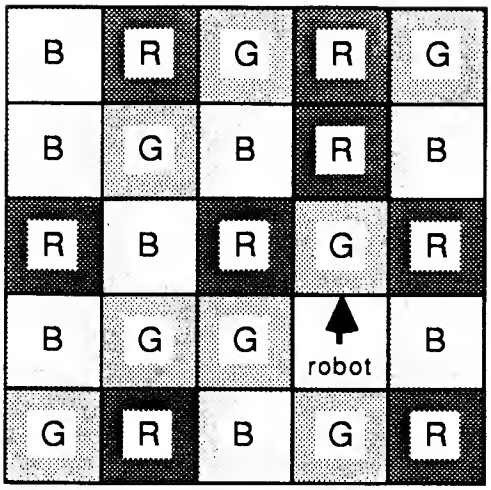

Figure 2.1: The 5 × 5 Grid World

2.8 Two Example Environments

The motivation for the introduction of the notion of diversity was the realization that many interesting "robot environments" can be modeled as finite automata which, although they have a large number of states, have low diversity. In this section, we make this point explicit by describing two particular small "robot environments".

2.8.1 $n \times n$ Grid World

Consider a robot on an $n \times n$ square grid (with "wraparound", so that it is topologically a torus). See Figure 2.1. The robot is on one of the squares and is facing in one of the four possible directions. Each square is either red, green, or blue. The robot can sense the color of the square it is facing. (This corresponds to the predicates of our previous development.)

The following actions are available to the robot: It can paint the square it faces red, green, or blue. The robot can turn left or right by 90 degrees, or step forward one square in the direction it is facing. Stepping ahead has the curious side effect of causing the square it previously occupied to be painted the color of the square it has just moved to, so moving around causes the coloring to get scrambled up.

This environment is a finite-state automaton which, even after reducing by factoring out some obvious symmetries, has an exponentially large (3^{n^2-1}) number of states.

However, the *diversity* of this environment is only $O(n^2)$. The state of this environment is completely characterized by knowing the color of each square (using a robot-relative coordinate system). It is not hard to devise a set of $O(n^2)$ tests whose results give all the

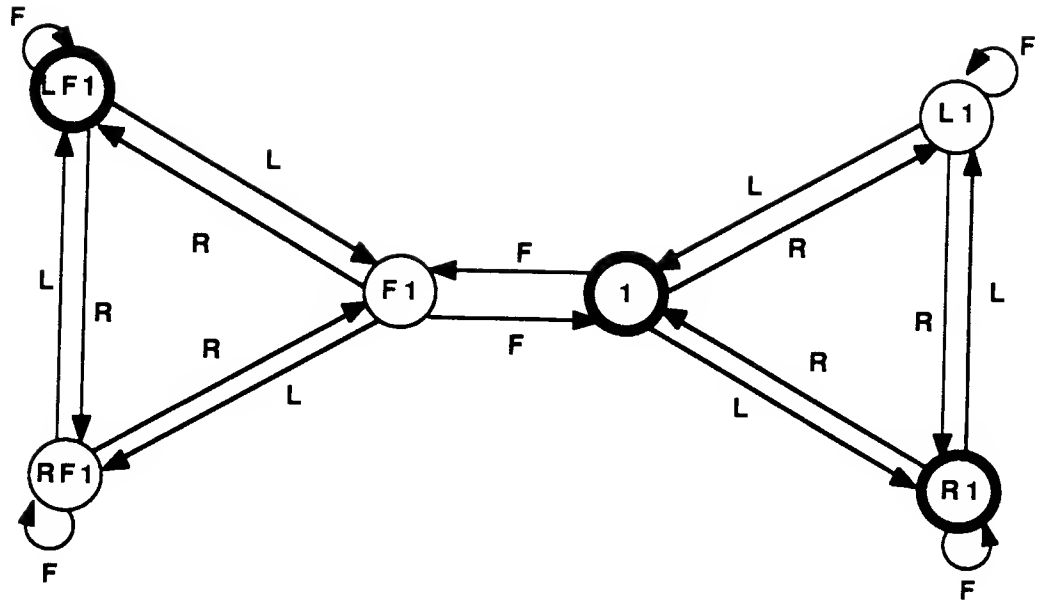

Figure 2.2: Update Graph of 3-bit Register World

desired information. (For example, the square behind the robot is red if and only if the test "`turn-left turn-left see-red`" is true.)

Given this information, it is easy to see how to predict the state of the environment after a given sequence of actions. In fact, it becomes clear that this is the "natural" representation of this environment, and that the intuitive representation and simulation procedure one would use for this environment are captured almost exactly by the diversity-based representation and simulation procedure given in the previous section.

We note that because of the "paint" operations, this environment is not a permutation environment.

2.8.2 n-bit Register World

In this environment, the robot is able to read the leftmost bit of an n-bit register. Its actions allow it to rotate the register left or right (with wraparound) or to flip the bit it sees.

Clearly, this automaton consists of 2^n global states, but its diversity is only $2n$ since there is one test for each bit, and one for the complement of each bit. We note that the register world is a permutation automaton.

The update graph of this environment is depicted in Figure 2.2. The name "1" in the figure refers to the predicate which returns **true** if the leftmost bit is a 1, and "L", "R" and "F" refer to the actions which rotate left and right, and which flip the leftmost bit. In the current state, the register contains the values 101. The borders of the tests which are

true in the current state have been deduced.

Chapter 3

Our Inference Procedure

The inference procedure tries to construct a perfect model of its environment by meeting the two requirements of the simulation theorem (Theorem 3). That is, the procedure first infers the structure of the update graph, and then maneuvers itself into a state q where it knows the value qt for every equivalence class $[t]$.

We will see that the first problem of constructing the update graph is by far the harder of the two. We therefore begin with the second problem of determining the associated value of each test equivalence class.

3.1 Inferring the Values of the Test Equivalence Classes

Suppose then that the update graph's structure is entirely known, and we now wish to determine the value associated with each vertex (equivalence class) of the graph.

Assign to each vertex a variable x_i which will stand for the value of that vertex in the starting state. Since the execution of any action causes each vertex to be updated with the value of one of the other vertices, we see that the value of each vertex in every future state will just be one of these variables x_i. Our goal is to reach a state in which all of the variables still in existence are known. (Some variables may disappear, but this is of no consequence since, for perfect predictability, we only need to know the values of those that still exist.)

Initially, all of the variables are unknown. We can "solve" for a particular variable x_i by causing one of the predicates p to be updated with the value x_i. In this state, x_i is the value of p which is directly observable.

If all of the existing variables are known, then we are done. Otherwise, there must be a vertex $[t]$, where $t = ap$, with unknown value x_i. Then by executing action a, we move the value of t to predicate p, and thus we learn the value of variable x_i. Repeating this process, we solve for all existing variables.

Note that the executed action sequence a above need not be longer than the size of the update graph $D(\mathcal{E})$. Further, each iteration of this loop decreases the number of unknown variables by one. Since there are initially only $D(\mathcal{E})$ variables, we see that this part of the inference problem can be solved in $O(D(\mathcal{E})^2)$ time.

We focus for the remainder of this chapter on the problem of inferring the structure of the update graph.

3.2 An Inference Procedure Using an Oracle for Equivalence

We begin by supposing that we have an oracle available that can tell us whether two tests s and t are equivalent.

Our algorithm (Figure 3.1) builds up the update graph, adding one edge at a time and creating new vertices when necessary, until no more edges can be added. Here, the program variable V represents the current set of vertices (equivalence classes). We assume that the predicates are inequivalent to one another, so initially V consists of one equivalence class for each of the predicates.

The edges of the graph are represented by the function χ: For each equivalence class $[t]$, and each basic action b, the program computes the vertex at the tail of the unique b-edge directed into $[t]$, so that $\chi([t], b) = [bt]$. If this is a vertex already in V, then an edge is simply added; otherwise, a new vertex $[bt]$ is first created and added to V before noting the new edge.

Since $|\mathtt{V}|$ is bounded by $D(\mathcal{E})$, we see that the procedure must halt, and in particular, makes no more than

$$|B||\mathtt{V}|^2 \leq |B|D(\mathcal{E})^2$$

calls to the equivalence testing oracle.

3.3 Determining If Two Tests Are Equivalent

We now turn our attention to the problem of determining whether or not two tests are equivalent. The inference procedure can *prove* that tests s and t are inequivalent if it can find a state q such that $qs \neq qt$; a single counterexample to the conjecture $s \equiv t$ suffices.

We wish to experiment with the available automaton $\mathcal{E}$ in order to prove $s \not\equiv t$. There are two problems we face:

- *(Accessibility of Counterexamples)* It may be difficult or impossible to get the automaton into a state q where $qs \neq qt$, even if such states exist.
- *(Irreversibility of Actions)* Even if we can get the automaton into such a state q, once we run test s we are in general unable to "back up" so as to be able to run test t.

Let us define two tests to be *compatible* if the action sequence of one is a prefix of the action sequence of the other. We note that irreversibility of actions is not a problem when

Input:
  P - set of predicates
  B - set of basic actions
  Oracle for testing if $s \equiv t$ for any tests s and t
Output:
  $\mathtt{V}$ - set of equivalence classes
  $\chi : \mathtt{V} \times B \to \mathtt{V}$ such that $\chi([t], b) = [bt]$
Procedure:
  $\mathtt{V} \leftarrow \{[p] \mid p \in P\}$
  while $\chi([t], b)$ is undefined for some $[t] \in \mathtt{V}, b \in B$ **do**
    if $bt \equiv s$ for some $[s] \in \mathtt{V}$ **then**
      $\chi([t], b) \leftarrow [s]$
    else
      $\mathtt{V} \leftarrow \mathtt{V} \cup \{[bt]\}$
      $\chi([t], b) \leftarrow [bt]$
    endif
  end

Figure 3.1: An Inference Algorithm Using an Oracle for Equivalence of Tests.

testing the equivalence of two compatible tests since they can be executed simultaneously. In particular, a predicate is compatible with all other tests.

We present solutions to these difficulties for the special class of permutation environments, and then discuss progress toward a solution in the general case.

3.4 Determining Test Equivalence in Permutation Environments

Assume then that $\mathcal{E}$ is a permutation environment. It is easy to show that each action permutes not only the global states, but the set of test equivalence classes as well. That is,

$$(\forall t \in T)(\forall s \in T)(\forall b \in B) s \equiv t \Leftrightarrow bs \equiv bt. \tag{3.1}$$

3.4.1 Overcoming Irreversibility of Actions

We show first how the problem of irreversibility of actions can be overcome by modifying the control structure of the basic algorithm so that any test can effectively be made compatible to any other test (Figure 3.2). This is essentially the same algorithm as in Figure 3.1; every new equivalence class is being compared against (nearly) all the known equivalence classes. However, the *order* in which these comparisons are made has been altered to ensure that every test in $\mathtt{V}$ can later be made compatible to any other test.

Input:
  P - set of predicates
  B - set of basic actions
  Oracle for testing if $s \equiv t$ for any tests s and t
Output:
  $\mathtt{V}$ - set of equivalence classes
  $\chi : \mathtt{V} \times B \rightarrow \mathtt{V}$ such that $\chi([t], b) = [bt]$
Procedure:
  $\mathtt{V} \leftarrow \{[p] \mid p \in P\}$
  while $\chi([t], b)$ is undefined for some $[t] \in \mathtt{V}, b \in B$ **do**
    $n \leftarrow 1$
    while $(\forall [s] \in \mathtt{V}) b^n t \not\equiv s$ **do**
      $n \leftarrow n + 1$
    for $1 \leq i < n$
      $\mathtt{V} \leftarrow \mathtt{V} \cup \{[b^i t]\}$
      $\chi([b^{i-1}t], b) \leftarrow [b^i t]$
    $\chi([b^{n-1}t], b) \leftarrow [s]$ {where $s \equiv b^n t$ and $[s] \in \mathtt{V}$}
  end

Figure 3.2: A Modified Inference Algorithm for Permutation Environments

The following theorem shows that no equivalence class is added twice to $\mathtt{V}$ by this algorithm, and furthermore that the inner loop is guaranteed to halt:

Theorem 7 *Let $[t]$ be a vertex in the program variable $\mathtt{V}$, b a basic action in B, and n a positive integer such that for all $[s] \in \mathtt{V}$ and all $1 \leq i < n$ we have $s \not\equiv b^i t$. Then the tests $bt, b^2t, \ldots b^{n-1}t$ are pairwise inequivalent.*

Proof: Suppose to the contrary that $b^i t \equiv b^j t$ for some i, j, $1 \leq i < j < n$. Then by (3.1), $t \equiv b^{j-i}t$ contradicting the hypothesis since $1 \leq j - i < n$ but $[t] \in \mathtt{V}$. ■

Essentially, the preceding theorem shows that the modified algorithm of Figure 3.2 is "just as good" as that of Figure 3.1 in the sense that both will correctly infer the update graph in roughly the same number of calls to the equivalence testing subroutine. Both algorithms also share the property that, at all times, the value of any equivalence class $[t]$ in $\mathtt{V}$ can be "read" directly simply by executing t. That is, if $t = ap, a \in A, p \in P$, then by executing a, we pass the current value of t to the predicate p where it can be observed directly.

The following theorem shows that the modified version of the algorithm has the additional property that the value of any $[t]$ in $\mathtt{V}$ can be not only "read," but "set up" as well. The theorem states that a path a can always be found in the current state of the update graph from some predicate class $[p]$ to $[t]$. Thus, by executing a, we pass the observable

value of $[p]$ to $[t]$. This property is crucial to the equivalence testing subroutine presented below.

Theorem 8 *Between each iteration of the outer loop of Figure 3.2, if* $[t]$ *is any vertex in* V *then a path exists in the current state of the update graph from some predicate's equivalence class to* $[t]$.

Proof: By induction on the number of iterations of the outer loop.

Initially, V consists only of predicate equivalence classes, and so the property holds trivially.

Suppose the theorem's statement holds at the top of one iteration of the loop. Consider the end of this iteration. We need to show there is a path from some predicate to each new $[b^i t], 1 \leq i < n$, added to V. We have $b^n t \equiv s$, for some $[s] \in$ V, and therefore, by the inductive hypothesis, we know of some $a \in A, p \in P$ for which a is a path from $[p]$ to $[s]$. Thus, $p \equiv as \equiv ab^n t = (ab^{n-i})b^i t$. In other words, ab^{n-i} is a path to $[b^i t]$ from the predicate equivalence class $[p]$. ■

Theorem 8 is used by the equivalence testing subroutine below. Although this procedure could be generalized for testing the equivalence of any two tests t and s, we assume here that the equivalence class of one of the tests, s, is already represented by a vertex $[s]$ in V. Then there is a path a from some predicate equivalence class $[p]$ to $[s]$; that is, $p \equiv as$. By (3.1) then, $t \equiv s$ if and only if $at \equiv as \equiv p$. Note that p, being a predicate, is compatible to at, and so the values of the two tests in a given state can be compared directly by executing both simultaneously.

Here is the algorithm for testing if s and t are equivalent:

1. Find a path a in the update graph from some predicate's equivalence class $[p]$ to $[s]$.
2. Get the environment into some random state q.
3. Execute p and at (simultaneously) to find their values in q: If $qp \neq qat$, then $s \not\equiv t$.
4. Repeat steps 2 and 3 until confident that $s \equiv t$.

Thus, we have overcome the problem of irreversibility of actions in permutation environments by applying knowledge already gathered about the structure of the update graph to effectively force the compatibility of any two tests which we might be interested in comparing for equivalence. Still missing from this algorithm are a method of effectively randomizing the environment (step 2), and a corresponding bound on the number of iterations of steps 2 and 3 necessary to confidently conclude that $s \equiv t$.

3.4.2 Overcoming Accessibility of Counterexamples

To rigorously prove that two tests are equivalent, we would have to show that their values are the same at each of the global states. In general, this is infeasible (one reason being that the state space may be enormous). Essentially, the preceding algorithm overcomes this difficulty by selecting a random sample from the state space. If at a single state the tests have different values, then the inference procedure may conclude with absolute certainty that the tests are inequivalent. Otherwise, the procedure concludes, with some possibility of error, that the tests are equivalent. We show below how this probability of error can be made vanishingly small. We prove that, in permutation environments, we have an adequate chance of finding a state in which the values of two inequivalent tests differ simply by taking an appropriate random walk.

We begin with a general discussion of random walks on directed graphs and of certain properties of point symmetric graphs (defined below), and next apply these results in proving a probabilistic upper bound on the running time of our algorithm.

3.4.2.1 Random Walks on Directed Graphs

We are concerned with random walks on a strongly connected (every vertex reachable from every other vertex) directed graph G which has n vertices and which is regular of degree d in the sense that every vertex has in-degree and out-degree equal to d. G may have self-loops and multiple edges between vertices. Let $A = \{a_{ij}\}$ denote the adjacency matrix of G, so that a_{ij} is the number of edges between vertex i and vertex j.

The random walk we are concerned with has the following form. We begin at an arbitrary vertex. At each step we first flip a fair coin. If we see "heads" then we stay at the current vertex, otherwise we pick one of the d outgoing edges uniformly at random and traverse it.

This random walk defines a finite Markov chain with transition matrix

$$B = \frac{1}{2}(I + \frac{1}{d}A). \tag{3.2}$$

If we let $\mathbf{p}_t$ denote the vector whose i-th component $\mathbf{p}_{ti}$ is the probability of the Markov chain being in state i (i.e. at vertex i) at time t, then we have the recurrence:

$$\mathbf{p}_{t+1}^T = \mathbf{p}_t^T B. \tag{3.3}$$

The initial vector $\mathbf{p}_0$ describes the probability of picking each vertex as the starting vertex.

We observe that the matrix B^m contains all positive entries for some positive integer m. Thus by the Perron-Frobenius theorem, B has an eigenvalue $\lambda_1 = 1$ with multiplicity 1 and corresponding eigenvector π. For any other eigenvalue λ of B, $|\lambda| < 1$. Also, it is

easy to see that since G is regular, the eigenvector $\pi = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$. This is the stationary distribution for our Markov chain.

As we take more and more steps in our random walk, the probability vector $\mathbf{p}_t$ converges to π; we lose track of where we began and are more or less equally likely to be at any vertex.

Theorem 9 *If $t = cdn^2$, then*

$$||\mathbf{p}_t - \pi|| \leq e^{-2c} \tag{3.4}$$

where $||x||$ is the ordinary Euclidean norm.

Proof: Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of B, where $\lambda_1 = 1$ and the other eigenvalues are arbitrary complex numbers arranged in order so that

$$1 = \lambda_1 > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|. \tag{3.5}$$

(We note that if λ is an eigenvalue of B, then so is $\bar{\lambda}$, since B is real.)

We now argue that the theorem follows if it can be shown that the maximum magnitude of any of $\lambda_2, \ldots, \lambda_n$ is bounded above by $1 - \frac{2}{dn^2}$.

Indeed, if we let $\mathbf{q}_t = \mathbf{p}_t - \pi$ denote the "error vector" at time t, then it follows that

$$||\mathbf{q}_{t+1}|| \leq |\lambda_2| \cdot ||\mathbf{q}_t||. \tag{3.6}$$

(This follows, for example, from the algebraic treatment of finite Markov chains given in [6].)

Since $||\mathbf{q}_0|| \leq 1$, it follows from our assumption that $|\lambda_2| \leq 1 - \frac{2}{dn^2}$ that

$$||\mathbf{q}_t|| \leq (1 - \frac{2}{dn^2})^t \leq e^{\frac{-2t}{dn^2}}. \tag{3.7}$$

This is at most e^{-2c} if $t = cdn^2$, as desired.

We now proceed to show that the maximum magnitude among $\lambda_2, \ldots, \lambda_n$ is at most $1 - \frac{2}{dn^2}$.

Let $\kappa_1, \ldots, \kappa_n$ denote the eigenvalues of the matrix $\frac{1}{d}A$, where $\kappa_1 = 1$. By the Perron-Frobenius theorem, all of the κ_j lie on or within the unit circle. Assuming that the indices for the κ_j's have been chosen appropriately, it follows from equation (3.2) that

$$\lambda_j = \frac{1}{2} + \frac{1}{2}\kappa_j \quad \text{for } j = 1, \ldots, n. \tag{3.8}$$

Therefore, all of the λ_j's lie within the circle in the complex plane with center at $\frac{1}{2}$ and radius $\frac{1}{2}$.

We begin with a result due to Fiedler [7] that applies to any doubly stochastic matrix—in our case the matrix $\frac{1}{d}A$. (This is a combination of his Lemma 3.5 and his Theorem 3.2.)

Define an eigenvalue κ of $\frac{1}{d}A$ to be "nonstochastic" if $\kappa \neq 1$. Fiedler's result says that the real part of any nonstochastic eigenvalue κ of $\frac{1}{d}A$ is bounded above:

$$\mathrm{Re}(\kappa) \leq 1 - 2(1 - \cos(\frac{\pi}{n}))\mu(S) \tag{3.9}$$

where $S = \frac{1}{2}(\frac{1}{d}A^T + \frac{1}{d}A)$ and $\mu(S)$ is defined by

$$\mu(S) = \min_{\emptyset \neq X \subsetneq V} \sum_{i \in X, j \in V - X} s_{ij}, \tag{3.10}$$

where $V = \{1, \ldots, n\}$. Here $\mu(S)$ is a "measure of the irreducibility" of S, S can be interpreted as the adjacency matrix for a graph H which is the average of the graph G and its inverse, and $\mu(S)$ is the minimum (over all partitions of the vertex set V into nonempty parts X and $V - X$) of the sum of the weights of the edges going from X into $V - X$.

In our case we can only say that

$$\mu(S) \geq \frac{1}{d}, \tag{3.11}$$

since all we know is that A is strongly connected. Thus Fiedler's theorem implies that

$$\mathrm{Re}(\kappa) \leq 1 - 2(1 - \cos(\frac{\pi}{n}))\frac{1}{d}. \tag{3.12}$$

Since $\cos(\frac{\pi}{n}) \leq 1 - \frac{4}{n^2}$ for $n \geq 2$, we have

$$\mathrm{Re}(\kappa) \leq 1 - \frac{8}{dn^2}. \tag{3.13}$$

It now follows that if λ is a "nonstochastic" eigenvalue of B, then λ must lie in the shaded region of the complex plane shown in Figure 3.3: From equation (3.8) and the fact that $|\kappa| < 1$, we see that λ must lie inside the circle C in the figure. Furthermore, combining equations (3.13) and (3.8), we see that the real part of λ is bounded above, and so λ must lie to the left of some line L. Thus, applying some elementary trigonometry, we obtain

$$|\lambda| \leq \sqrt{1 - \frac{4}{dn^2}} \leq 1 - \frac{2}{dn^2}. \tag{3.14}$$

■

If we set c to be approximately $\log(n)$, we have the following easy corollary:

Corollary 1 *After $t = dn^2 \log(n)$ steps we have a chance of at least $\frac{1}{2n}$ of being at any given vertex.*

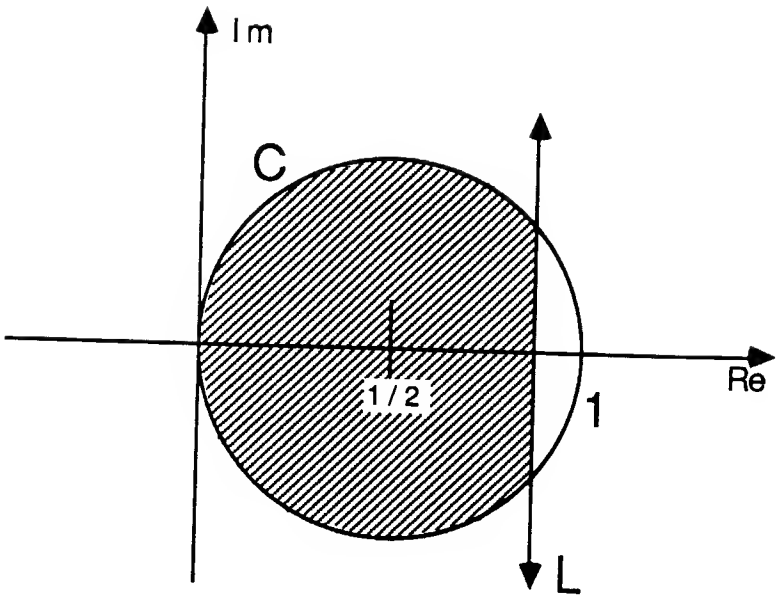

Figure 3.3: Region of Complex Plane in Which Nonstochastic Eigenvalues May Lie

3.4.2.2 Point Symmetric Graphs

Next, we turn to a discussion of point symmetric graphs, and prove a lemma needed in proving Theorem 10 below.

Definition 2 *A graph G is* point symmetric *if for all pairs of vertices v, w in G, there exists an automorphism on G which maps v to w.*

Definition 3 *A bipartite graph G is* bipartite point symmetric *if for all pairs of vertices v, w on the same side of the graph, there exists an automorphism on G which maps v to w.*

It is easy to see that all vertices have the same degree in a point symmetric graph, and likewise for all vertices on the same side of a bipartite point symmetric graph.

The proof of the following lemma is due in large part to Satish Rao:

Lemma 1 *Let $G = (V, E)$ be an undirected, connected point symmetric or bipartite point symmetric graph with degree at least d at every vertex. Let m be the minimum number of edges that must be removed to separate G into two non-empty pieces. Then $m \geq d$.*

Proof: For arbitrary subsets S, T of vertices, let $D(S,T)$ be the number of edges connecting points in S with points in T, and let $C(S)$ be the number of edges cut in separating S from the rest of the graph:

$$D(S,T) = |\{\{s,t\} \in E \mid s \in S, t \in T\}|.$$

$$C(S) = D(S, V - S).$$

Then $m = \min\{C(S) \mid \emptyset \neq S \subsetneq V\}$.

Suppose, to the contrary of the theorem's statement, that $m < d$, and let S be the smallest non-empty subset of V for which $C(S) = m$.

Since $C(S) > 0$, S contains some *boundary point* j, that is, a vertex j connected to some vertex outside of S.

We claim S contains an *interior point* i as well, i.e., a vertex not on the boundary. If this were not the case, then all $k = |S|$ vertices in S are boundary points so that $k \leq m$. The number of edges connecting vertices in S is at least

$$\begin{aligned} \frac{dk - m}{2} &> \frac{dk - d}{2} \\ &= \frac{d(k-1)}{2} \\ &> \frac{k(k-1)}{2} \end{aligned}$$

Clearly, it is impossible for more than $\binom{k}{2}$ edges to connect k points.

In the case that G is only bipartite point symmetric, we can assume that i and j are on the same side of the graph. Suppose otherwise. Then the k_1 vertices on one side of the graph are interior, and the k_2 vertices on the other side are boundary points. Thus $k_2 \leq m$, and so the number of interior edges is at most $k_1 k_2 \leq k_1 m < k_1 d$, a contradiction since the k_1 vertices on the first side are interior.

Therefore, in either case, we may conclude that there is an automorphism σ on G mapping i to j. Let S' be the image of S under σ. Then $|S| = |S'|$ and $C(S') = C(S) = m$. Since j is a boundary point of S but an interior point of S', $S \neq S'$.

Let $I = S \cap S', X = S - I, X' = S' - I$, and $Z = V - (S \cup S')$ (Figure 3.4). Since $j \in I$, I is not empty. The sets X and X' are also non-empty since S and S' are unequal sets of the same size. Therefore, $0 < |X| < |S|$ and so $C(X) > m$. Similarly, $C(X') > m$.

We have:

$$\begin{aligned} C(S) &= D(X, Z) + D(X, X') + D(I, X') + D(I, Z) \\ C(S') &= D(X', Z) + D(X', X) + D(I, X) + D(I, Z) \\ C(X) &= D(X, Z) + D(X, X') + D(X, I) \\ C(X') &= D(X', Z) + D(X', X) + D(X', I) \end{aligned}$$

Thus, we have the following contradiction:

$$\begin{aligned} 2m &= C(S) + C(S') \\ &= C(X) + C(X') + 2D(I, Z) \end{aligned}$$

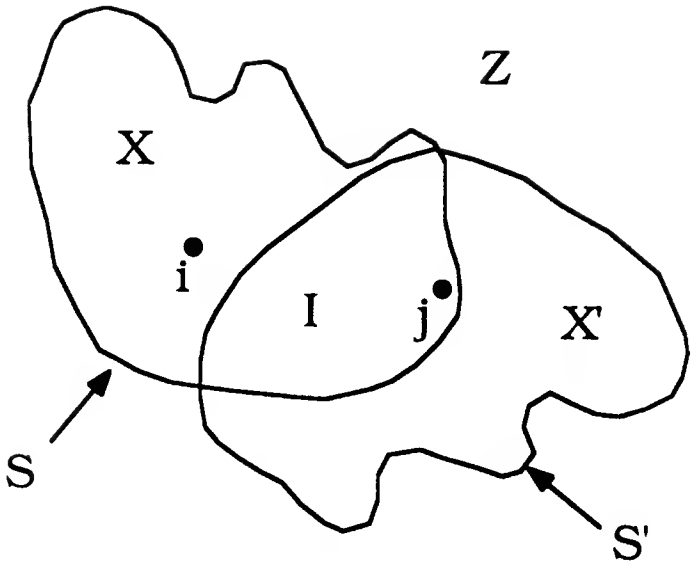

Figure 3.4: Construction for Lemma 1

$$\geq C(X) + C(X')$$
$$> 2m.$$

■

3.4.2.3 Finding Counterexamples with Random Walks

With these results, we are finally able to prove:

Theorem 10 *Let s and t be two inequivalent tests of a permutation environment $\mathcal{E}$ of diversity D. We take a random walk of length $2|B|D^4\log(D)$ beginning at an arbitrary start state. At each step, with equal probability, we either do nothing, or we execute a uniformly and randomly chosen basic action from B. Then the probability that the values of s and t differ at the state where we complete this walk is at least $\frac{1}{2D}$.*

Proof: Consider the graph P defined as follows: The vertices of P are all ordered pairs $([as],[at])$ for all $a \in A$, and an edge b is directed from vertex $([s_1],[t_1])$ to $([s_2],[t_2])$ iff $s_1 \equiv bs_2$ and $t_1 \equiv bt_2$. Clearly, P has no more than $D(D-1) \leq D^2$ vertices. Further, as with the update graph, the vertices are permuted by each basic action, so there is exactly one ingoing and one outgoing edge for each basic action at each vertex. (Alternatively, P can be viewed as the left coset graph of the subgroup which stabilizes both $[s]$ and $[t]$.)

Let $a = b_1 \ldots b_n$ be the chosen random sequence of basic actions, and let q be the starting state. When a is executed, the environment moves to state qa where s and t have the values qas and qat. In other words, s and t are updated with the values of as and at in

state q. The tests as and at have different values at q if and only if s and t have different values at the completion of a.

Thus, we can regard the *reverse* of the random walk a as an equally random walk through P; at each step, we move from vertex $([b_{i+1} \ldots b_n s], [b_{i+1} \ldots b_n t])$ to $([b_i b_{i+1} \ldots b_n s], [b_i b_{i+1} \ldots b_n t])$ by traversing the reversed edge b_i, and finally arriving at $([as], [at])$.

Since we are taking a random walk of just the form and length described in the hypothesis of Corollary 1 for a graph such as P with at most D^2 vertices, and both indegree and outdegree equal to $|B|$ at each vertex, we see that our (reversed) random walk has a roughly equal chance of finishing at any of the vertices of P.

We now argue that, for at least $\frac{1}{D}$ of the vertices $([s'], [t'])$ of P, we have $qs' \neq qt'$. This, combined with the preceding arguments, will prove the lower bound on the probability of finding a counterexample.

Let the *orbit* of any test u be the set $O_u = \{[au] \mid a \in A\}$.

Consider the graph C defined as follows: The vertex set V of C is the union $O_s \cup O_t$, and an (unlabeled) edge is directed from $[s']$ to $[t']$ if $([s'], [t'])$ is a vertex of P—that is, if $s' \equiv as$ and $t' \equiv at$ for some action $a \in A$.

We argue first that C is (bipartite) point symmetric. If $[s_1], [s_2]$ are in O_s, then there is some action a for which $s_2 \equiv as_1$. Let σ be the permutation mapping each vertex $[u]$ to $[au]$. Then σ maps $[s_1]$ to $[s_2]$ and furthermore defines an automorphism on C since if $([s'], [t'])$ is an edge, then so are $([as'], [at'])$ and $([a^{-1}s'], [a^{-1}t'])$. Similarly, for any two tests in O_t, there is an automorphism on C mapping the first to the second.

By the definition of orbits, we have that O_s and O_t are either equal or disjoint. In the former case, the preceding argument shows that C is point symmetric. In the other case, C is a bipartite point symmetric graph.

In either case, let d_s be the outdegree of each vertex in O_s (necessarily the same at each vertex by the preceding argument) and similarly define d_t as the indegree of each vertex in O_t. Then the number of edges in C is exactly $d_s|O_s| = d_t|O_t|$. Let $d = \min\{d_s, d_t\}$.

Let X be the set of vertices $[u]$ of C for which qu is **true**. Then each edge connecting (in either direction) a vertex in X with another in its complement corresponds to a vertex $([s'], [t'])$ in P for which $qs' \neq qt'$. We therefore would like to show that at least $\frac{1}{D}$ of the edges of C connect X to its complement. This will be the case if we can find at least d such edges.

Since $s \not\equiv t$, there is at least one such edge. Let C' be the subcomponent of C connected to this edge. The graph C' is still (bipartite) point symmetric. Therefore, simply regarding the edges of C' as undirected, and applying Lemma 1 to it, we see that at least d edges are

cut in separating X from its complement in C, as desired.

This completes the theorem. ■

Using this result, we can show the following theorem, the main result of this section:

Theorem 11 *Let $\mathcal{E}$ be a permutation environment with diversity D. Given $\epsilon > 0$, our algorithm will infer the structure of $\mathcal{E}$ in time*

$$O(|B|^2 D^7 (\log(\frac{|B|D}{\epsilon}))(\log(D))) \tag{3.15}$$

with probability of error less than ϵ.

Proof: The preceding theorem states that the probability of distinguishing two inequivalent tests, having taken an appropriate random walk, is at least $\frac{1}{2D}$. Thus, the probability of failing to do so after n trials is no greater than $(1-\frac{1}{2D})^n$. This error probability is bounded by a parameter δ when

$$n \geq \frac{\log \delta}{\log(1-\frac{1}{2D})}.$$

As many as $I = |B|D^2$ inequivalence tests may be made in the course of inferring the automaton. The probability, then, of successfully distinguishing all of the inequivalent pairs of tests is at least $(1-\delta)^I$. Our goal is to make this probability more than $1-\epsilon$. We have been given ϵ and choose $\delta \leq \frac{\epsilon}{I}$. Then

$$1 - \epsilon \leq 1 - I\delta \leq (1-\delta)^I$$

as desired.

Finally, if we choose $n \geq 2D \log \frac{I}{\epsilon}$, then our probability of error on an individual experiment is sufficiently small since

$$\begin{aligned} 2D \log \frac{I}{\epsilon} &\geq \frac{\log \frac{I}{\epsilon}}{\log \frac{2D}{2D-1}} \\ &= \frac{\log \frac{\epsilon}{I}}{\log(1-\frac{1}{2D})} \\ &\geq \frac{\log \delta}{\log(1-\frac{1}{2D})}. \end{aligned}$$

Here, we have used the fact that $\log x \leq x - 1$ for all x. (In particular, if $x < 1$ then $\log x \leq x - 1 \Rightarrow \frac{1}{\log \frac{1}{x}} \leq \frac{1}{1-x}$. Above, we have applied this formula with $x = 1 - \frac{1}{2D}$.)

Hence, our procedure requires I inequivalence tests. Each of these requires up to $2D \log \frac{I}{\epsilon}$ experiments, each of which can involve a random walk of length $2|B|D^4 \log(D)$. (The time to run the actual experiment, or to determine which experiment is to be performed next is negligible.) We thus arrive at the running time stated in the theorem. ■

Thus we have completed our algorithm by exhibiting an effective random walk technique. Note that, implicitly, we have assumed that the diversity, or an upper bound D_{max} on the diversity, has been given to the inference procedure since the diversity must be known to calculate the length and number of random walks needed. If no such bound is available, the algorithm can be executed repeatedly with $D_{max} = 1, 2, 4, 8, \ldots$. If D_{max} is smaller than the true diversity D, then either the algorithm will be unable to build a small enough update graph, or it will construct an incorrect update graph which will sooner or later make a wrong prediction. When either of these occur, we double D_{max} and run the inference procedure again.

The bounds stated in the preceding theorems have been tightened significantly since our original presentation of the algorithm. Empirically, however, we have found that much shorter random walks and far fewer experiments are sufficient, and we therefore conjecture that the bounds are still not tight.

3.5 Determining Test Equivalence in General

We discuss now the general case in which $\mathcal{E}$ is not necessarily a permutation environment. We don't at the moment know how to handle in a rigorous manner the first difficulty of finding a state in which two inequivalent tests can be distinguished, even if we assume that $\mathcal{E}$ is strongly connected. Nonetheless, in practice this may often not be a concern; if two tests s and t are inequivalent then there are usually many easily reached states q such that $qs \neq qt$.

We now propose a technique for handling the irreversibility of actions in general environments.

We need to figure out how to get $\mathcal{E}$ into a state q where we *know* the value of the test qt, even though we haven't run test t yet, so that we can run test s instead.

Let $t = ap$; here a is the action part of test t and p is the predicate.

Suppose we run action a repeatedly. Eventually the predicate p will exhibit periodic behavior. Once we know that this periodic behavior has been established, and once we know the period m of this behavior, then we can figure out the value of qt for the current state q without having to run the test t.

We have to address the problem that for general finite-state automata, it is well known that the eventual period can be as large as $|Q|$, the size of the automaton. This would be a serious problem for our proposed approach, since the size can be an exponential function of the diversity. However, the following theorem shows that the period is no larger than the diversity.

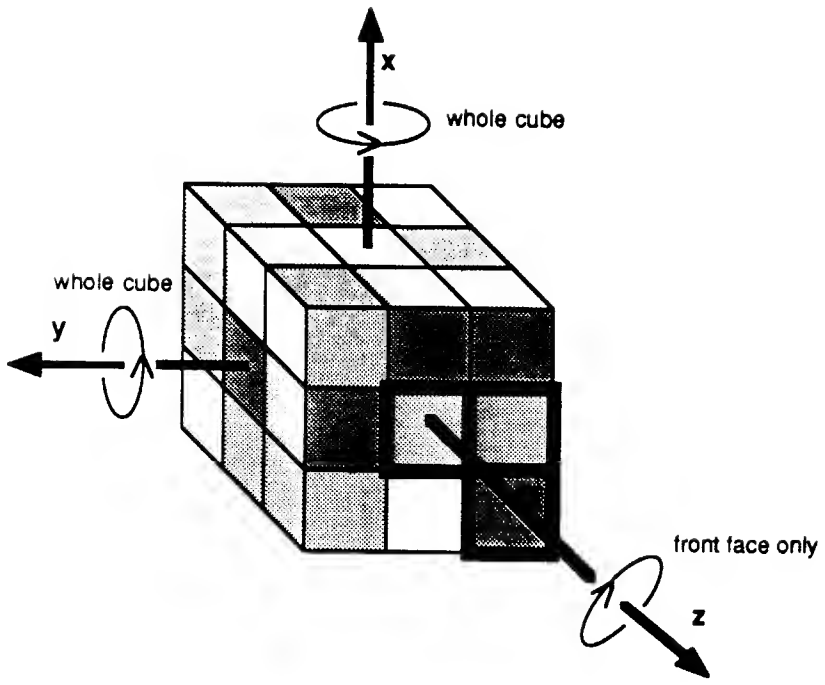

Figure 3.5: The Rubik's Cube World

Theorem 12 *Let $D = D(\mathcal{E})$. If we run action a repeatedly, then the behavior of predicate p will exhibit transient behavior for no more than D steps, and then will settle down into periodic behavior with period at most D.*

Proof: This follows easily from our simulation theorem (Theorem 3). Consider the sequence of tests $p, ap, a^2p, \ldots, a^Dp$. Since there are only D test equivalence classes, by the pigeon hole principal, at least two of these tests are equivalent. Say $a^ip \equiv a^jp$ where $i < j$. Recall that p is passed its value from a^kp under action a^k. Therefore, p will exhibit transient behavior for at most the first i executions of a, and will then settle into periodic behavior with period $j - i$. ■

To complete the description of our inference procedure, we suppose as above that an upper bound D_{max} is available on the diversity $D(\mathcal{E})$ of the automaton being inferred.

To run the algorithm of Figure 3.1, we need a way to test s and $t = ap$ for inequivalence. The following procedure is suggested by the previous theorem:

- Get the environment into some random state.
- Run action a for D_{max} steps. (This is to eliminate transient behavior of p.)
- Run action a for $2D_{max}$ steps, keeping track of qp for each state q reached.

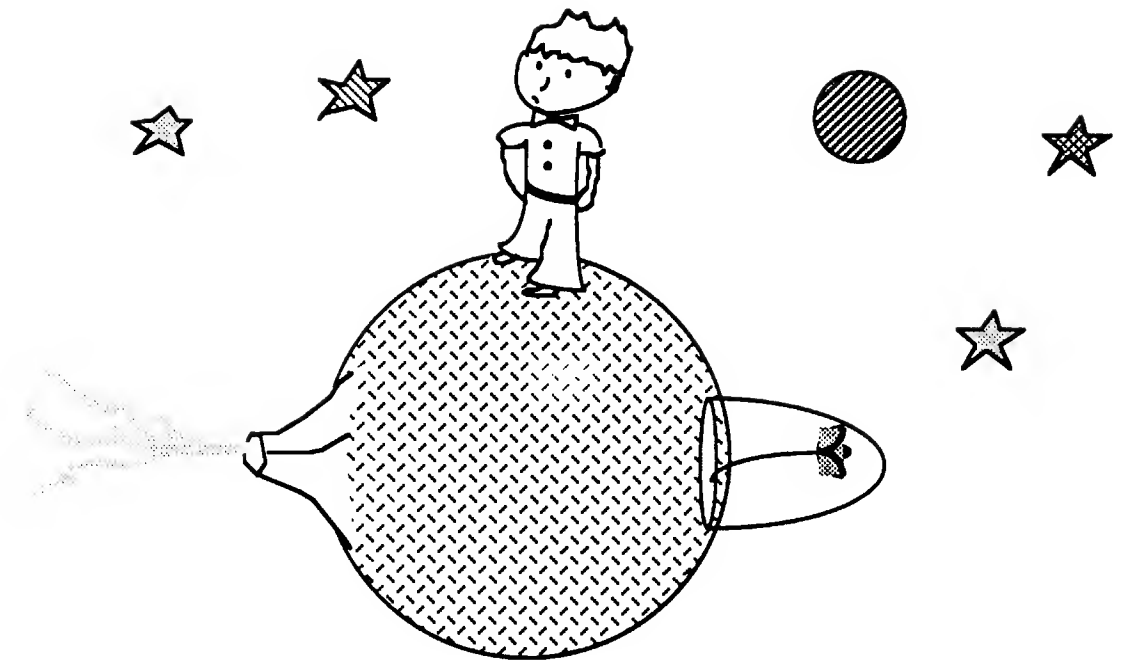

Figure 3.6: The Little Prince's Planet

- Use the information gathered in the previous step to determine the period of predicate p under action a. Use this information to determine whether qt is **true** or **false** in the current state q (without running test t).
- Run test s to determine qs.
- If $qs \neq qt$, then $s \not\equiv t$.
- Repeat until confident that $s \equiv t$.

As before, this is a one-sided test: a report that $s \not\equiv t$ is certainly correct, but a report that $s \equiv t$ may be erroneous.

The test must be re-run a number of times before concluding that $s \equiv t$. To make the trials as independent as possible, we may:

- Take a "random walk in $\mathcal{E}$" between each trial, by executing some randomly chosen sequence of actions.
- Repeatedly execute an action ab instead of just a in each trial, where b is an arbitrarily chosen action in A.

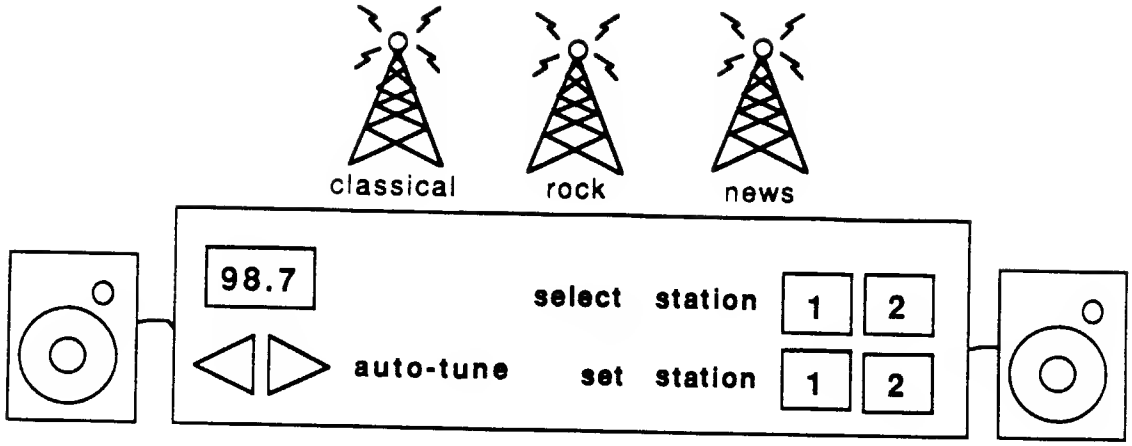

Figure 3.7: The Car Radio World

These heuristics may not help to find a counterexample in all cases; but are reasonably effective in practice. (We hope to prove the effectiveness of these techniques as we did in the permutation environment case for a broader class of finite automata.)

Also, for efficiency, we are in many instances able to force compatibilities as in the permutation environment case, and can often compare many tests against many other tests in single experiments. These heuristics lead to many-fold improvements of our running times.

3.6 Experimental Results

3.6.1 Three More Toy Environments

Consider the following permutation environment based on "Rubik's Cube" (Figure 3.5). The robot is allowed to see only three of the fifty-four tiles: a corner tile, an edge tile and a center tile, all on the front face. Each of these three senses can indicate any one of six colors. The robot may rotate the front face, and may turn the whole cube about the x and y axes. (By reorienting the cube he can thus turn and view any of the six faces.)

As another example environment, consider a robot just delivered to the "Little Prince" [16] on his home planet (an asteroid, really). This planet has a rose and a volcano, which the robot can see when he is next to them; the available sense values are "See Volcano" and "See Rose". The planet is very small—it takes only four steps to go all the way around it. The basic actions available to the robot are "Step Forward", "Step Backward", and "Turn Around". See Figure 3.6. In the state shown, the robot has no sensations, but he will see

Environment	Diversity	Global States	$\|B\|$	$\|P\|$	Version	Time	Moves	Senses	Experiments
Little Prince	4	4	3	2	P	0.1	303	102	51
					M	0.2	900	622	50
Car Radio	9	27	6	1	M	3.7	27,695	9,557	1,146
Grid World	27	$\approx 10^{11}$	6	1	M	90.4	583,195	123,371	9,403
Rubik's Cube	54	$\approx 10^{19}$	3	3	P	126.3	58,311	4,592	2,296
					M	401.3	188,405	79,008	2,874
32-bit Register	64	$\approx 10^{9}$	3	1	P	29.8	270,771	10,914	5,457
					M	18.3	52,436	29,884	300

Table 3.1: Experimental Results

the volcano if he takes a step forward, and will see the rose if he takes a step backwards (or turns around and takes a step forwards).

In the last micro-world, the robot can fiddle with the controls of a car radio (see Figure 3.7) and can detect what kind of music is being played. There are three distinctive stations which define the robot's sensations: rock, classical, and news. The robot can use the auto-tune to dial the next station to the left or right (with wrap-around), or can select one of the two programmed stations, or can set one of these two program buttons to the current station. Unlike the last two environments, the Car Radio World is not a permutation environment because of the robot's ability to program stations.

3.6.2 Summary of Results

Table 3.1 summarizes how our procedures handled these environments, as well as the 5×5 Grid World environment and the 32-bit Register environment described in Section 2.8.

The most complicated environment (Rubik's Cube) took less than two minutes of CPU time to master—we consider this very encouraging.

Rubik's Cube, the Little Prince and the 32-bit Register Worlds were explored with an implementation (version "P") which exploits the special properties of permutation environments, but which only compares one pair of tests at a time. All worlds were explored as well by version "M", which tries to compare many tests against many other tests in a single experiment. The run times given are in seconds. The last three columns give the number of basic actions taken by the robot, the number of sense values asked for, and the number of experiments performed. (An experiment is defined loosely as a sequence of actions and senses from which the robot deduces a conclusion about equivalence between tests. Information about several tests may be obtained in a single experiment, and the same sequence of actions and senses may be repeated several times, each repetition counting as one experiment. Also, we have generalized the notion of a test here to allow the function γ to map $Q \times P$ into an arbitrary set of sensations, not necessarily the set $\{\mathbf{true}, \mathbf{false}\}$. For

example, in the Grid World, a single predicate gives the color (red, green or blue) of the square faced by the robot.) These implementations were done in C on a DEC MicroVax II workstation.

Chapter 4

Inference of Visible Simple Assignment Automata with Planned Experiments

In this chapter, we focus on the problem of *planning* experiments when trying to infer the structure of a finite automaton by experimentation. In the preceding chapters, we were concerned with the same general problem. However, our focus was on the identification of *hidden state variables*, rather than on the planning of experiments.

The experimental technique used in the preceding chapters was a simple one based on the properties of random walks. As a consequence, we could only prove our techniques to be effective for a restricted class of automata (permutation automata). The key difficulty in extending our proof is that random walks are not in general guaranteed to get the automaton into a desired state (or set of states) with sufficiently high probability. For the general case, it seems clear that experiments have to be *planned* carefully.

This chapter does not address the issue of hidden state variables; *we assume that all state variables are visible to the observer.* We make this simplification to bring to the foreground the issues regarding the planning of experiments. Of course, at some point we would like to merge the techniques developed here with those for identifying hidden state variables.

Aside from this difference in the visibility of state variables, the automata we study are structurally identical to those studied up to this point. Recall from Section 2.6 that every finite-state deterministic system can be represented as a simple assignment automaton in which each variable stands for one test equivalence class. In this chapter, to simplify our discussion, we drop the equivalence class terminology, and instead formally redefine an environment as a simple assignment automaton.

4.1 Definitions

We define a *simple assignment automaton* to be a tuple (V, B, δ, q_0) such that

- $V = \{x_1, \ldots, x_n\}$ is a finite nonempty set of n binary *state variables*,
- B is a finite nonempty set of *input symbols*, also called *basic actions*,

- δ is a function from $\{1, \ldots, n\} \times B$ into $\{1, \ldots, n\}$; δ is called the *update function*, and
- q_0 (the *initial state* of the automaton) is a function mapping V into $\{0, 1\}$.

The (global) *state* of the automaton is an assignment of a binary value to each variable in V.

On input $a \in B$, the automaton makes a transition from its current state $\mathbf{x} = (x_1, \ldots, x_n)$ to the state $\mathbf{x}' = (x'_1, \ldots, x'_n)$ where

$$x'_i = x_{\delta(i,a)}; \tag{4.1}$$

each variable is updated by a simple assignment from the value of some other variable (or possibly the same variable).

As before, we let Q denote the set of all global states q reachable from the initial state q_0 of the automaton.

In Section 2.6 we argued that every finite-state binary output Moore automaton is equivalent to a simple assignment automaton where one or more of the state variables specifies the output. The number of state variables in the smallest corresponding simple assignment automaton is just the diversity of the original finite-state automaton.

We say that a simple assignment automaton is *visible* if all of its local state variables are observable.

We assume henceforth that we are dealing with a particular visible simple assignment automaton $\mathcal{E} = (V, B, \delta, q_0)$, which we call the *environment* of the learning procedure.

We assume that $\mathcal{E}$ is *reduced* in the sense that, for each pair of distinct variables $x_j, x_k \in V$, there is a state $q \in Q$ such that $x_j \neq x_k$ at q. (This assumption is made for simplicity here to avoid degenerate but easily handled cases where variables are indistinguishable.)

We let $A = B^*$ denote the set of all sequences of zero or more basic actions in the environment $\mathcal{E}$; A is the set of *actions* possible in the environment $\mathcal{E}$, including the *null action* λ.

We extend δ to the domain $\{1, \ldots, n\} \times A$ in the natural way: $\delta(i, \lambda) = i$ and $\delta(i, ba) = \delta(\delta(i, a), b)$ for $i \in \{1, \ldots, n\}, b \in B, a \in A$. Thus $\delta(i, a)$ identifies the variable whose value x_i takes under action a; equation (4.1) now holds for any $a \in A$.

Finally, we assume that $\mathcal{E}$ is *strongly connected*: it is possible to get from any state in Q to any other. (Otherwise, it may be impossible to infer $\mathcal{E}$ completely, since $\mathcal{E}$ will get stuck in one of its several strongly connected components.)

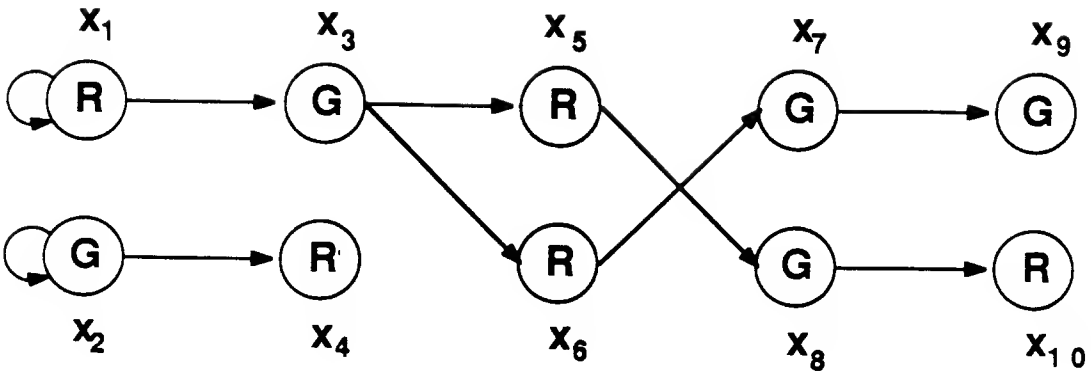

Figure 4.1: The Effect of Action p in Our Example Simple Assignment Automaton

4.2 Example

To make things concrete, consider the simple assignment automaton $\mathcal{E}$ illustrated in Figure 4.1.

Here $\mathcal{E}$ has n binary state variables $\{x_1, \ldots, x_n\}$, where n is even. We think of the values of these variables as being drawn from the set $\{Red, Green\}$.

We imagine the n variables as being divided into $n/2$ "columns", where x_{2i-1} and x_{2i} are in the same column, for $i = 1, \ldots, n/2$.

There are four input symbols, or "basic actions": p, q, r, s. On any input, the variables in the i-th column are updated in some way from the variables in the $i-1$st column. (We assume that the variables in the first column never change value—x_1 is always *Red* and x_2 is always *Green*.) Since each of x_{2i-1} and x_{2i} can be assigned one of x_{2i-3} or x_{2i-2} in two ways, there are a total of four distinct ways in which the variables in column i can depend upon those in column $i-1$. Each input symbol is associated with one of these possibilities, but in a manner that is *arbitrary and varies from column to column.* Figure 4.1 illustrates the effect of action p, and a typical state of the automaton; the other three actions could be illustrated with similar diagrams.

It is important to note that two of the four possibilities are guaranteed to give a column a monotone coloration, independent of whether the column to the left has a monotone or a mixed coloration.

This automaton has a number of states which is exponential in n — it is easy to see that every column except the first can be made all *Red* or all *Green*. And there are many

other states where columns other than the first have a mixed coloration.

However, it is easy to see that in order for a column to receive a mixed coloration, its neighbor to the left must have had a mixed coloration on the previous step. Furthermore, mixed colorations are easily destroyed as the column colorations move rightwards. Once a column has a monotone coloration, this coloration propagates to the right unchanged with each input. It should be clear that a random string of input will have a small chance of giving a mixed coloration to any columns except a few of the leftmost ones.

We now observe that in order for an inference algorithm to figure out how the later columns are wired together, the algorithm *must* propagate the mixed colorations all the way down to the right. This can only be accomplished by careful planning and execution of experiments, and not by random walk techniques.

We view this example as a fancy kind of "combination lock", since the algorithm must figure out a correct "combination" for giving column $i-1$ a mixed coloration before it can figure out a correct combination for column i. (Of course, there are many correct combinations, but there are many more incorrect ones.)

It is not too hard to figure out how to approach this particular example, given all of the "side information" stated above. However, we must remember that the inference algorithm we seek is only told that it is to infer a simple assignment automaton where all local state variables are visible — it is *not* told such things as that the variables are paired up into columns, each column is updated from the one to the left, etc. In the absence of such side information, the general problem can be challenging.

4.3 Our Inference Procedure

We now present a procedure for inferring $\mathcal{E}$ by systematic experimentation. Our procedure is given as input V, B, and the ability to experiment with $\mathcal{E}$ by executing basic actions (i.e. giving the automaton inputs) and observing the state changes. Our procedure outputs the unknown function δ, in time polynomial in $n = |V|$ and $|B|$.

The algorithm maintains, as its fundamental data structure, a *candidate set* $C(i,b)$ of possible values for the update function $\delta(i,b)$, for each variable x_i and each $b \in B$. Initially $C(i,b) = V$ for all i and b.

Our basic strategy is to repeatedly plan and execute experiments which cause at least one $C(i,b)$ to shrink. When no such experiment is possible $C(i,b) = \{\delta(i,b)\}$ for all i and b, so that δ has been identified.

Definition 4 *We say $b \in B$ is an* immediately useful experiment *if there exist i, j, k such that j and k are both in $C(i,b)$, and $x_j \neq x_k$.*

If we execute the immediately useful experiment b then either j or k is removed from $C(i,b)$ (e.g. j is removed if the new value for x_i differs from the old value for x_j).

Finding an immediately useful experiment (if one exists) is easy since it requires knowledge of C but not of δ. But what shall we do if there are no immediately useful experiments to do?

In such a case, there may exist some "setup action" $a \in A$ that will make $b \in B$ an immediately useful experiment. We call the combined action ab a "useful experiment".

Definition 5 *Let $\sigma = ab$ where $a \in A, b \in B$. We call σ a* useful experiment *if there exist i,j,k such that $x_{\delta(j,a)} \neq x_{\delta(k,a)}$ and j and k are both in $C(i,b)$.*

The trouble with this notion is that to tell if ab is a useful experiment requires knowing the unknown function δ, in order to predict the effect of setup action a. We need an *effective* way of finding useful experiments.

We introduce the notion of a "plausible experiment" to remedy this defect.

First, as with the function δ, we extend C to the domain $\{1,\ldots,n\} \times A$: $C(i,\lambda) = \{i\}$ and $C(i,ba) = \bigcup_{j \in C(i,a)} C(j,b)$ for $i \in \{1,\ldots,n\}, a \in A, b \in B$.

Definition 6 *We call $\sigma \in A$ a* plausible experiment *if there exist i,j,k such that j and k are both in $C(i,\sigma)$, and $x_j \neq x_k$.*

Knowledge of C, but not δ, is all that is required to find plausible experiments.

Note that all useful experiments are plausible since $\delta(i,a) \in C(i,a)$ always. However, not all plausible experiments are useful. Our inference procedure depends on the following critical theorem.

Theorem 13 *The* shortest *plausible experiment is also the shortest useful experiment.*

Proof:

Because every useful experiment is plausible, we need only show that the shortest plausible experiment is useful.

Let $\sigma = ab, a \in A, b \in B$ be the shortest plausible experiment. Let j,k be members of $C(i,\sigma)$ for which $x_j \neq x_k$. Then there exist $r,s \in C(i,b)$ for which $j \in C(r,a)$ and $k \in C(s,a)$. Since σ is the shortest plausible experiment, and because $|a| < |\sigma|$, all the variables in $C(r,a)$ must have the same value. In particular, $x_{\delta(r,a)} = x_j$, and likewise, $x_{\delta(s,a)} = x_k$. Therefore $x_{\delta(r,a)} \neq x_{\delta(s,a)}$, so that σ is useful. ■

Not only is the shortest plausible experiment useful, but there always exists a plausible experiment up until the point when the inference task is finished.

Theorem 14 *If there exists an i and b such that $|C(i,b)| > 1$, then there exists a plausible experiment (and thus a shortest plausible experiment).*

Proof: Let x_r and x_s be two distinct variables in $C(i,b)$. By assumption, there exists a global state q for which x_r and x_s obtain differing values, and such a state q is reachable from the current state (via some action a). Then $\sigma = ab$ is a useful (and therefore plausible) experiment. ■

4.3.1 The Basic Inference Algorithm

We now give a high-level description of our inference procedure, assuming the availability of a subroutine which plans the shortest useful experiment.

Initially, each $C(i,b) = V$. Our procedure then repeatedly finds and executes useful experiments, each of which eliminates at least one variable from at least one candidate set.

How many experiments are performed before each candidate set is a singleton? Since there are $|B|n$ candidate sets, each initially of size n, at most $|B|n^2$ experiments are performed. The following theorem gives a tighter bound.

Theorem 15 *After no more than $|B|n$ useful experiments are performed, each candidate set will be a singleton set.*

Proof: An easy induction shows that, between each experiment, for fixed $b \in B$, two candidate sets $C(i,b)$ and $C(j,b)$ must either be disjoint or identical. (Two such sets will be identical if and only if $x_i = x_j$ in every global state seen so far. When a state is first observed for which $x_i \neq x_j$, the common set $C(i,b) = C(j,b)$ is split into two disjoint nonempty blocks, one of which becomes the new $C(i,b)$ and one of which becomes the new $C(j,b)$.) Thus each set $C(i,b)$ is a block of a partition S_b of a subset of V into pairwise-disjoint, non-empty subsets. Initially, $S_b = \{V\}$; there is only one block. Each useful experiment ending in b causes at least one set $C(i,b)$ to shrink, and so causes one or more of the blocks in S_b to either split or shrink. After n such operations, each block of S_b (and therefore each candidate set $C(i,b)$ as well) will be a singleton. Thus, at most n experiments are performed ending in each of the $|B|$ basic actions. ■

The proof of this theorem suggests an efficient representation of the candidate sets. Rather than storing the sets explicitly, we maintain the partition S_b, and represent each $C(i,b)$ as a pointer to one of the blocks in S_b. This allows faster updating of the candidate sets between each experiment.

Figure 4.2 gives a high-level description of our procedure (less the assumed experiment planning subroutine PLAN-EXP).

Input: V, B, and access to the environment $\mathcal{E} = (V, B, \delta, q_0)$.
Output: δ
Procedure:
for $b \in B$
$S_b \leftarrow \{V\}$
for $i \in \{1, \ldots, n\}$: $C(i, b) \leftarrow V$.
while PLAN-EXP can find a useful experiment $\sigma = ab$ **do**
Execute a. Let $(x_1, \ldots, x_n)$ be the resulting state.
Execute b. Let $(x'_1, \ldots, x'_n)$ be the resulting state.
for $s \in S_b$
Let $\pi(s, 0) = \{i \in s \mid x_i = 0\}$.
Let $\pi(s, 1) = \{i \in s \mid x_i = 1\}$.
for $i \in \{1, \ldots, n\}$: $C(i, b) \leftarrow \pi(C(i, b), x'_i)$
$S_b \leftarrow \bigcup_{i \in \{1, \ldots, n\}} \{C(i, b)\}$
for $i \in \{1, \ldots, n\}, b \in B$
Output "$\delta(i, b) = x$", where $C(i, b) = \{x\}$.

Figure 4.2: The Basic Inference Algorithm

Observe that each step of the main **while** loop takes $O(n)$ time, except possibly for the execution of the experiment returned by PLAN-EXP whose length we discuss below.

4.3.2 The Experiment Planning Subroutine

The subroutine PLAN-EXP is given the candidate sets and the current state, and is asked to find the shortest useful experiment. By Theorem 13, this experiment is also the shortest plausible experiment.

We can find the shortest plausible experiment by searching the space of unordered pairs of variables $\{j, k\}$, both in some set $C(i, \sigma)$, until we find one for which $x_j \neq x_k$. More precisely, we do a breadth-first search of the forest of trees in which the root of each search tree is a pair $\{i, i\}$, and the b-children of each node $\{j, k\}$ are the pairs $\{j', k'\}$ for which $j' \in C(j, b), k' \in C(k, b)$. When a pair $\{j, k\}$ is found for which $x_j \neq x_k$, we return the experiment which is the path from the node $\{j, k\}$ to the root of its tree.

Since we search a forest of $O(n^2)$ vertices, each of degree $O(|B|n^2)$, this experiment planning subroutine runs in time $O(|B|n^4)$. Furthermore, the length of the experiment returned is bounded by the size of the search space, n^2. Thus, the entire inference algorithm will run in time $O(|B|^2 n^5)$, having executed $|B|n^3$ basic actions.

We now improve these bounds with a more efficient subroutine (Figure 4.3) which maintains equivalence classes of variables using a "weighted union and collapsing find" data structure. Initially, all the elements of each candidate set (or, equivalently, of each parti-

Input: $C(i,b)$ for $i \in \{1,\ldots,n\}$, $b \in B$, and $x_1,\ldots,x_n$
Output: a useful experiment σ
Procedure:
  for $i \in \{1,\ldots,n\}$: Place i in an equivalence class by itself.
  for $b \in B, s \in S_b$
    Let j be an arbitrary member of s.
    $J \leftarrow$ FIND(j)
    for $k \in s - \{j\}$
      $K \leftarrow$ FIND(k)
      if $J \neq K$ **then**
        $J \leftarrow$ UNION(J,K)
        enqueue $(\{j,k\},b)$
  while queue not empty **do**
    dequeue $(\{j,k\},\sigma)$
    if $x_j \neq x_k$ **then return** σ
    for $b \in B$
      let j' be an arbitrary member of $C(j,b)$
      let k' be an arbitrary member of $C(k,b)$
      $J \leftarrow$ FIND(j'), $K \leftarrow$ FIND(k')
      if $J \neq K$ **then**
        UNION(J,K)
        enqueue $(\{j',k'\},b\sigma)$
  return FAIL

Figure 4.3: The Experiment Planning Subroutine PLAN-EXP

tion block) are merged into the same equivalence class. To merge a pair $\{j,k\}$, we check that the two are in the same equivalence class; if they are not, their equivalence classes are UNIONed and the pair is placed on a queue. Thus, a UNION operation is always coupled with an addition to the queue. When the pair $\{j,k\}$ is dequeued, the members of $C(j,b)$ are merged with those of $C(k,b)$ for all the basic actions b, and the process continues.

The subroutine is constructed so that if $(\{j,k\},\sigma)$ is on the queue, then $j,k \in C(i,\sigma)$ for some i. Thus, if $x_j \neq x_k$, then σ is a plausible experiment.

During the execution of the subroutine, if $(\{j,k\},\sigma)$ was the last pair enqueued, then the current *search depth* is defined to be $|\sigma|$. It is clear that the search depth increases incrementally.

The next theorem is useful in analyzing and seeing the correctness of the subroutine.

Theorem 16 *Suppose $j,k \in C(i,\sigma)$. Then the subroutine of Figure 4.3 (if not interrupted to return an answer) will merge j and k into the same equivalence class before the search depth exceeds $|\sigma|$.*

Proof: By induction on $|\sigma|$.

If $|\sigma| = 1$, then $j, k \in C(i, b)$ for some $b \in B$, and j and k are merged into the same equivalence class during the initialization phase when the search depth is exactly one.

Let $h > 1$ and suppose that the theorem's statement holds when $|\sigma| < h$. Given $j, k \in C(i, \sigma)$, where $|\sigma| = h$, we wish to show that j and k are merged before the search depth exceeds h.

Let $\sigma = ba, b \in B, a \in A$ and let r, s be such that $r, s \in C(i, a)$ and $j \in C(r, b), k \in C(s, b)$. Since $|a| = h - 1$, r and s have been merged by the time the search depth reaches h, by our inductive hypothesis. Thus, there must have been a series of UNION operations performed to bring this about. Since each UNION operation is coupled with an addition to the queue, there must have been a series of enqueuings of the form:

$$\begin{aligned} (\{r = r_0, r_1\} &, \sigma_0) \\ (\{r_1, r_2\} &, \sigma_1) \\ (\{r_2, r_3\} &, \sigma_2) \\ &\vdots \\ (\{r_m, r_{m+1} = s\} &, \sigma_m). \end{aligned}$$

When $(\{r_x, r_{x+1}\}, \sigma_x)$ is dequeued, the members of the candidate sets $C(r_x, b)$ and $C(r_{x+1}, b)$ are merged into one equivalence class, so that, transitively, the sets $C(r, b)$ and $C(s, b)$ are merged into one. In particular, j and k's equivalence classes are merged. Since each $|\sigma_x| < h$, this happens before the search depth exceeds h. ■

Corollary 2 *The first plausible experiment discovered by the subroutine (i.e. the one returned) will also be the shortest plausible experiment.*

Corollary 3 *If there exists a plausible experiment, then the subroutine will discover it. That is, a return of FAIL by the procedure will be correct.*

Clearly, the running time of the procedure is bounded by the number of UNION-FIND operations. Since we begin with n equivalence classes, no more than n UNIONs can be performed. Therefore, n bounds the total number of enqueuings, and so the search depth as well. Based on this fact and the fact that S_b is a partition of at most n elements, we see that $O(|B|n)$ FIND operations are performed, yielding a running time for the subroutine of $O(|B|n \cdot \alpha(|B|n))$, where α is an extremely slow growing functional inverse of Ackerman's function. (See [17].) Finally, the length of the experiment constructed cannot exceed the maximum search depth of n. Thus, we have:

Theorem 17 *Our inference algorithm correctly infers the environment $\mathcal{E}$ in time $O(|B|^2 n^2 \alpha(|B|n))$, having executed no more than $|B|n^2$ basic actions.*

4.4 Optimality

In this section, we prove that the upper bound on the number of basic actions executed by our inference algorithm is (within a constant factor of) the best possible.

Theorem 18 *There exists a constant $\epsilon > 0$ such that, for all $n \geq 4, m \geq 3$, there exists a simple assignment automaton $\mathcal{E}$ for which $|B| = m$ and $|V| = n$, and which cannot be inferred by any algorithm which executes fewer than $\epsilon|B|n^2$ basic actions.*

Proof: Consider the following "combination lock" environment $\mathcal{E}$, similar to the example described in Section 4.2: $n = |V| \geq 4, |B| \geq 3$. B contains a special "clear" symbol c. The "lock's combination" is the sequence $a_1 a_2 \ldots a_{n-2}$ where $a_1 = c$ and $a_i \in B - \{c\}$ for $1 < i < n-1$. The update function δ is defined as follows:

- $\delta(1, b) = 1$ for $b \in B$
- $\delta(n, b) = n$ for $b \in B$
- $\delta(i, a_{i-1}) = i - 1$ for $1 < i < n$
- $\delta(i, b) = n$ for $1 < i < n, b \in B - \{a_{i-1}\}$.

Initially, only x_1 is true.

It is easy to verify that x_1 is always true, x_n is always false, and no more than one variable at a time (other than x_1) can be true. If $1 < i < n$, the variable x_i will be true if and only if the action sequence $a_1 a_2 \ldots a_{i-1}$ was just executed.

Consider the set P of pairs (i, b) where $2 < i < n, b \in B - \{c\}$ and $\delta(i, b) = n$ (i.e., $b \neq a_{i-1}$). To positively identify $\mathcal{E}$, an inference algorithm must, for each such pair in P, eliminate the possibility that $\delta(i, b) = i - 1$. It is not hard to see that the only experiment which will do this is the sequence $\sigma_{i,b} = c a_2 a_3 \ldots a_{i-2} b$. Let $E = \{\sigma_{i,b} \mid (i, b) \in P\}$. Clearly, $|E| = |P|$. At some time, each experiment in E must be executed; however, no two of these experiments can overlap by our construction. Thus, the number of basic actions executed must be at least

$$\sum_{\sigma \in E} |\sigma| = \sum_{2<i<n} (|B| - 2)(i - 1) = \Omega(|B|n^2).$$

■

Chapter 5

Conclusions and Open Problems

We have presented a new representation for finite-state systems (environments), and proposed a new procedure for inferring a finite state environment from its input/output behavior.

In the case of permutation environments, our procedure can infer the structure of the environment in expected time polynomial in the diversity of the environment, and $\log(\frac{1}{\epsilon})$, where ϵ is an arbitrary positive upper bound given on the probability that our procedure will return an incorrect result.

For general environments, our procedure appears to work well in practice, although we don't have a proof to this effect.

When the environment has lots of "structure", the diversity will typically be many orders of magnitude smaller than the number of global states of the environment; in these cases our procedure can offer many orders of magnitude improvement in running time over previous methods.

Finally, we have shown how to infer any visible simple assignment automaton in time polynomial in the number of variables and basic actions in that automaton, and have shown that our procedure is optimal to within a constant factor in terms of the number of basic actions executed.

Future work should be directed toward methods of handling, or handling better, a broader class of environments. Environments apparently not handled well by our current techniques include those with:

- Actions with conditional effects (such as a Grid World with boundaries, so that the "step ahead" action has no effect *if* the robot is facing and up against the boundary).
- Dependence on global state variables or control variables (e.g. an "on-off switch in the Car Radio World).
- States which are difficult to reach (consider the "combination lock" environment of Chapter 4 which is almost always in a locked state, and is unlikely to be unlocked by trying random combinations).

- Actions with probabilistic effects (such as a "spin" operator in the Grid World, which leaves the robot facing in a random direction).
- Actions or sensations which are subject to noise, and so may have unreliable effects or be providing unreliable information.
- Environments which are infinitely large (such as an infinitely long Register World).

The question of how to apply the planning techniques of the last chapter to the general problem of inferring automata with hidden variables remains open. Also open is the question of what other classes of automata can be inferred by techniques similar to those used for inference of permutation environments. Finally, what other models of learning (such as mistake bound learning as in [13]) can be applied to the problem of inference of finite automata?

Bibliography

[1] Dana Angluin. Inference of reversible languages. *Journal of the ACM*, 29(3):741–765, July 1982.

[2] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[3] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[4] Dana Angluin and Carl H. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[5] Gary L. Drescher. *Genetic AI – Translating Piaget into Lisp.* Technical Report 890, MIT Artificial Intelligence Laboratory, February 1986.

[6] William Feller. *An Introduction to Probability and its Applications.* Volume 1, John Wiley and Sons, third edition, 1968.

[7] Miroslav Fiedler. Bounds for eigenvalues of doubly stochastic matrices. *Linear Algebra and its Applications*, 5(3):299–310, July 1972.

[8] E. Mark Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[9] E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[10] E. Mark Gold. System identification via state characterization. *Automatica*, 8:621–636, 1972.

[11] J. Hartmanis and R. E. Stearns. *Algebraic Structure Theory of Sequential Machines.* Prentice-Hall, 1966.

[12] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw-Hill, second edition, 1978.

[13] Nick Littlestone. Learning when irrelevant attributes abound. In *Proceedings of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 68–77, October 1987.

[14] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceeding of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, October 1987.

[15] Ronald L. Rivest and Robert E. Schapire. A new approach to unsupervised learning in deterministic environments. In Pat Langley, editor, *Proceeding of the Fourth International Workshop on Machine Learning*, pages 364–375, Irvine, California, June 1987.

[16] Antoine de Saint-Exupéry. *The Little Prince*. Harcourt, Brace, & World, 1943.

[17] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975.